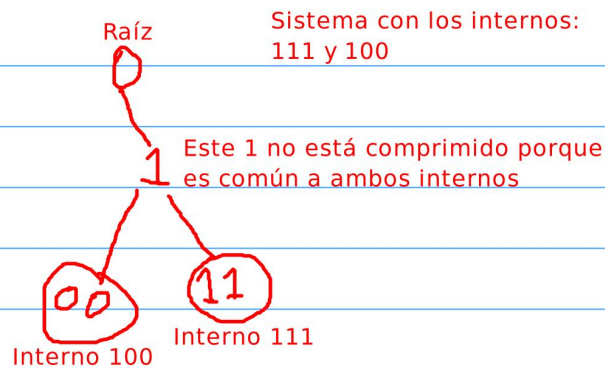


```

1 b)
2 string es vector<digito>
3 digito es nat
4
5 CentralCeroInfinita se explica con TAD CentralCeroInfinita
6 genero: central
7 central se representa con estr, donde estr es:
8 tuplac
9   internos: conjTrieComprimido(string), // Ver más abajo la explicación
10  usuarios: diccAVL(nat, <pTrie: itConjTrieComprimido(string, nat), pConj: itConjAVL(nat)> ),
11  usr: conjAVL(nat) // conjunto de usuarios
12 >
13
14 - internos es un conjunto que contiene cadenas de internos.
15 - usr es un conjunto de idUsuarios.
16 - usuarios es un diccionario de nro. de usuario a tupla con punteros a los elementos correspondientes en las otras componentes de la estructura.
17
18 b) OPERACIONES:
19
20 AgregarInterno(inout sistema: central, in interno: string, in usuario:idUsuario)
21   1) Definir en sistema.internos a interno //O(L)
22   2) Agregar en sistema.usr al usuario // O(Log(n))
23   --> Las operaciones anteriores devuelven iteradores a las estructuras, p1 y p2 respectivamente
24   3) Definir en sistema.usuarios al usuario con <p1,p2> como significado // O(Log(n))
25
26
27 EliminarUsuario( inout sistema: central, in usuario: idUsuario )
28   1) Obtener de sistema.usuarios los dos punteros a las otras estructuras (p1 y p2) // O(Log(n))
29   2) Usar ambos punteros para eliminar de las respectivas estructuras al usuario // O(Log(n) + 1)
30   3) Eliminar de sistema.usuarios al interno especificado // O(Log(n))
31
32
33 ExisteInterno?(in sistema: central, in interno: string) → res: bool
34   -Fijarse si el interno esta incluido en sistema.internos. Si lo está, retornar true. //O(L)
35
36
37 Interno(in sistema: central, in usuario:idUsuario) → interno: string
38   -Buscar en sistema.usuarios el significado de idUsuario y obtener el iterador (p) al elemento correspondiente en sistema.internos //O(Log(n))
39   -Retornar una referencia no modificable de SiguienteClave(p)
40
41
42 Usuarios(in sistema: central) → usrs: conjunto(idUsuario)
43   -Devuelve una referencia no modificable de sistema.usr //O(1)
44
45
46 PuedeAgregarse?(in sistema: central, in interno: string) → res: bool
47
48 Para efectuar esta operacion en la complejidad pedida, la implementación de conjTrieComprimido que estamos utilizando deberá proveer la operación
49
50 [ GeneraPrefijo?(in dic: conjTrieComprimido(string), in elem: string) → res: bool ]
51
52 (ver explicacion en la imagen adjunta)
53
54 c) ALGORITMOS
55
56 AgregarInterno(inout sistema: central, in interno: string, in usuario:idUsuario): //interpretamos a la secuencia de digitos como un string
57
58   it1 := conjTrieComprimido.Definir(sistema.internos, interno, usuario); // O(1)
59   it2 := ConjAVL.Agregar(sistema.usr, usuario); // O(log(n))
60   DiccAVL.Definir(sistema.usuarios, usuario, <it1, it2>); // O(L)
61
62 EliminarUsuario(inout sistema: central, in usuario: idUsuario)
63
64   <it1, it2> := DiccAVL.Obtener(sistema.usuarios, usuario); //O(Log(n))
65   IteradorDiccTrieComprimido.EliminarSiguiente(it1) // O(1)
66   IteradorConjAVL.EliminarSiguiente(it2) // O(Log(n))
67   DiccAVL.Eliminar(sistema.usuario, usuario); // O(Log(n))
68
69
70 d) Invariante de representacion
71 i) No puede haber prefijos en las claves de e.internos
72
73 ii) Para todo < u , <p1,p2> > de e.usuarios:
74   - tiene que haber siguiente de p2 , luego u = Siguiente(p2):
75
76
77 iii) Las claves de e.usuarios debe ser igual a e.usr
78
79 iv) Para todo par de usuarios distintos, se cumple que hay siguiente de sus p2 asociados y cada uno se corresponde con un interno
80   distinto presente en e.internos.
81
82 v) El cardinal de e.internos y e.usuarios es el mismo
83
84 Rep(e) = i) ∧ ii) ∧ iii) ∧ iv) ∧ v)
85
86 i) (Va,b: string)(a ∈ Claves(e.internos) ∧ b ∈ Claves(b.internos) ∧ a ≠ b ⇒L noEsPrefijo(a,b) )
87
88 noEsPrefijo(a,b) ≡ (len(a) > len(b)) ∨L (Vi: nat)(0 < i < len(b) ⇒L (∃j: nat)(0 < j ≤ i ∧L a[j] ≠ b[j])) )
89
90 ii) (Vu: idUsuario)(Definido?(e.usuarios, u) ⇒L (haySiguiente(Obtener(e.usuarios, u).pTrie) ∧ haySiguiente(Obtener(e.usuarios, u).pConj) ∧L
91   u = Siguiente(Obtener(e.usuarios, u).pConj) ) )
92
93 iii) Claves(e.usuarios) = e.usr
94
95 iv) (Va,b: usuario)(Def?(e.usuarios, a) ∧ Def?(e.usuarios, b) ⇒L
96   haySiguiente(Obtener(e.usuarios, a).pTrie) ∧ haySiguiente(Obtener(e.usuarios, b).pTrie) ∧L
97   Def?(e.internos, Siguiente(Obtener(e.usuarios, a).pTrie)) ∧ Def?(e.internos, Siguiente(Obtener(e.usuarios, b).pTrie)) ∧
98   Siguiente(Obtener(e.usuarios, a).pTrie) ≠ Siguiente(Obtener(e.usuarios, b).pTrie))
99
100 v) Cardinal(e.internos) = #Claves(e.usuarios)
101
102
103 e) funcion de abstracción
104
105 (Vc: central) Abs(e) = c |
106   e.usr = obs usuarios(c) ∧L
107   (Vu: usuario)( u ∈ e.usr ⇒L Siguiente(Obtener(e.usuarios, u).pTrie) =obs interno(c, u))
108
109
110 f) (ver explicacion en la imagen adjunta)
111
112
113
114
115
116

```



Cada interno tiene en el último nodo todos los dígitos que no sean comunes a ningún otro interno del sistema. Así, para eliminar un interno, simplemente elimino ese último nodo, ya que todos los dígitos anteriores tienen que quedarse en el trie porque son parte de otro interno del sistema.

Para puedeAgregarse? recibo un interno  $i$ , y lo que tengo que hacer es recorrer el trie siguiendo ese interno  $i$  (metiéndome también en los nodos comprimidos). Si en algún momento mientras recorro siguiendo  $i$  me encuentro con un interno ya definido, entonces no puedo agregar a  $i$ , porque este interno sería prefijo de  $i$ . Si llegué al final de  $i$ , y todos sus dígitos ya estaban definidos en el trie, tampoco puedo agregarlo, porque existe algún interno en el sistema tal que  $i$  es prefijo suyo. Si no sucede ninguna de estas cosas, entonces puedo agregar a  $i$ .

Para saturado:

Pensemos en una instancia de una estructura de representación de un trie comprimido. Llamamos:

Nodo comprimido: nodo que contiene más de un carácter.

Nodo hoja: todo nodo del trie donde finaliza una búsqueda.

Nodo saturado: todo nodo del trie en el cual la búsqueda podría continuar por cualquier letra del alfabeto (0 ó 1).

Para implementar Saturado? en  $O(1)$ , deberíamos contar con un `DiccTrieComprimido` que nos permita saber cuántos nodos de cada tipo hay. Esto puede mantenerse actualizado en las operaciones de Agregar y Eliminar elementos del conjunto. Notamos que como la estructura es libre de prefijos, no se puede insertar ningún elemento que extienda la búsqueda a partir de un nodo hoja (ya que haría que alguna palabra previamente definida pase a ser prefijo de la nueva palabra) y tampoco se puede insertar ningún elemento cuya búsqueda termine en un nodo ya definido (ya que sería prefijo de alguna palabra pre-existente). Sí se pueden insertar palabras que extiendan a nodos no saturados (a través de un carácter que no tengan definido) o palabras que hagan que se 'divida' un nodo comprimido. Por lo tanto, se puede agregar una palabra nueva si solo si existe algún nodo comprimido o algún nodo no saturado.